Project 3 Lab Report

Ann Keenan
akeenan2

Environment

The processes were run on a MacBook Air, Mid 2012 model with the following specifications.

- macOS High Sierra v. 10.13.3
- 1.8 GHz Intel Core i5 processor
- 4GB RAM
- Intel HD Graphics 4000 1536 MB
- 480 GB Macintosh SSD card

`Mandelmovie` runs with the following default call to `mandel`:

```
./mandel -x 0.2910234 -y -0.0164365 -s <zoom> -m 1000 -W 700 -H 700 -o <filename> -n <num_threads>
```

The zoom is betweeen 2 and 0.00001, the filename is `mandel%d.bmp` with the integer value being between 1 and 50. The number of threads is by default set to 1. The program takes 4.6 seconds to complete at a zoom of 0.00001.

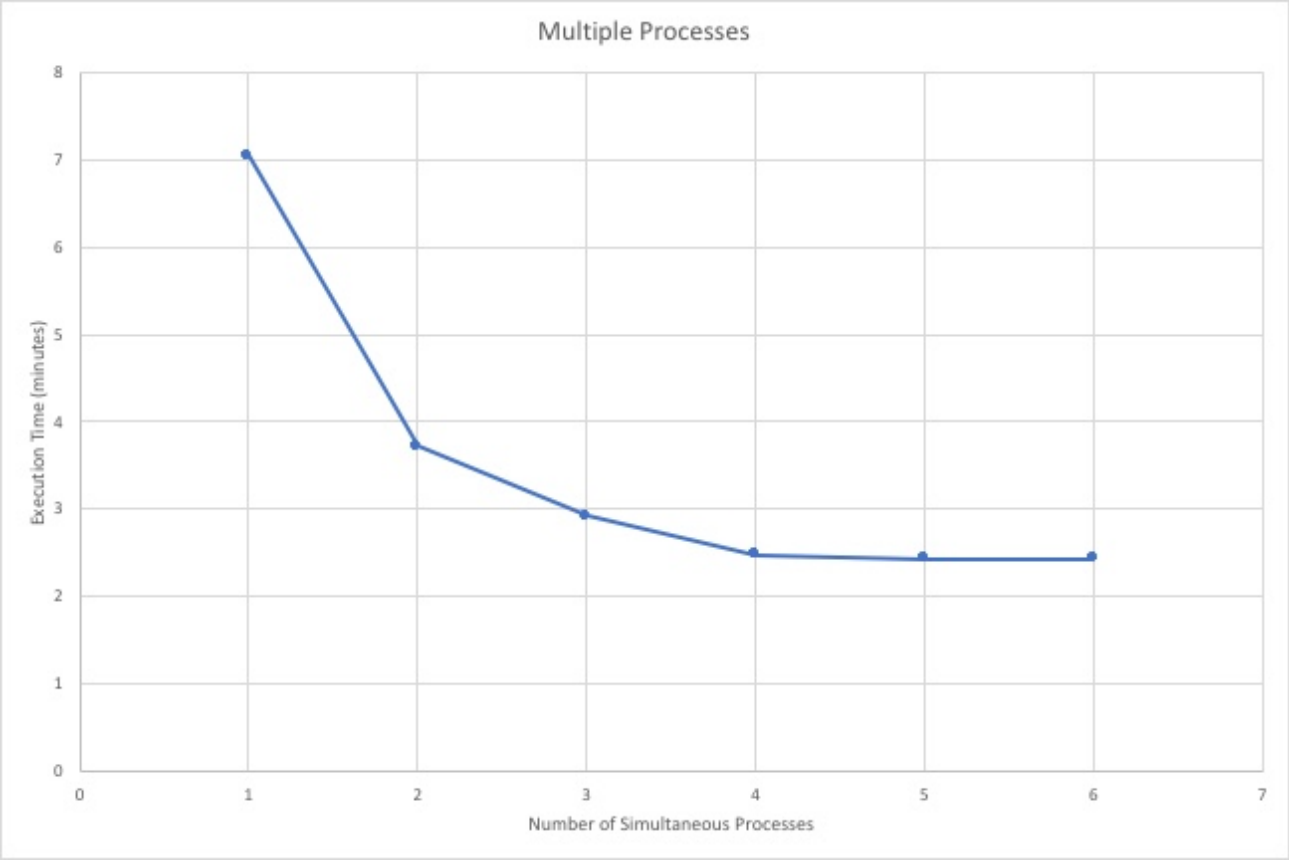
Mandelmovie

Experimental Setup

`Mandelmovie` was run with a range of 1 to 6 processes running simultaneously, with the runtime for each measured with the unix `time` command. The commands run were `time ./mandelmovie [0-6]`, and the `real` time outputted by the `time` command was used to form the graphs and conclusions.

Graph

Processes	Time (m-s)	Time (m)
1	7m2.715s	7.04525
2	3m42.869s	3.71448
3	2m55.361s	2.92268
4	2m28.533s	2.47555
5	2m25.940s	2.43233
6	2m25.362s	2.42270



Discussion

The performance greatly increased from one process running at a time to two processes running simultaneously, decreasing in runtime by more than 4-1/2 minutes. From two to three processes running simultaneously, the decrease in time to complete all processes was less sharp, at a little under a minute, and from three to four it was even less at under 30 seconds of a decrease in runtime. Running 4, 5, and 6 simultaneous processes had about the same performance, finishing within 3 seconds of one another. This points to a trend that adding more simultaneous processes would not greatly decrease the total completion time of the program.

Somewhere between 3 and 4 processes seems to be the optimal number of simultaneous processes, as the program did not run much quicker when more processes were added. Therefore, adding more simultaneous processes makes the program run quicker up to the point where hardware constrains the amount of computational power allowed.

Depending on the question being asked, it is both not possible and possible to have too many processes. In terms of the program itself, the input is constrained to being between 1 and 50, as only 50 images are being generated and there would be no point in having more than 50 processes. However, running the program with more than 4 processes does not speed up executing time, so in this sense, it would be an overload on the hardware to try to run all 50 processes at once.

Mandel

Experimental Setup

The multithreaded mandel program was run using 1, 2, 4, 8, and 12 threads, with the execution time measured by the `time` command. Each measurement was repeated five times with the fasted time out of the five used for the graph analysis.

Graph

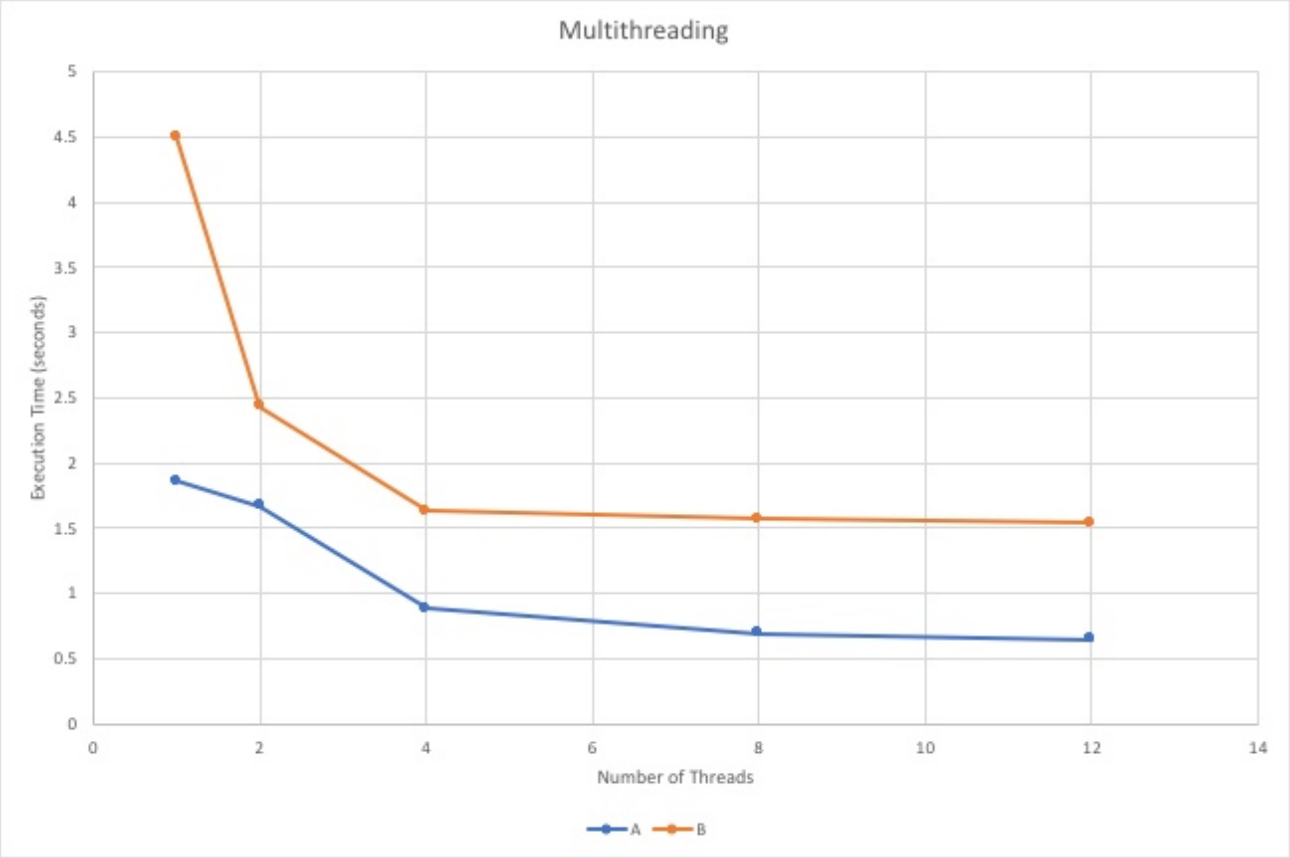
A:

Threads	Time (m-s)	Time (s)
1	0m1.858s	1.858
2	0m1.672s	1.672
4	0m0.883s	0.883

8	0m0.693s	0.693
12	0m0.652s	0.652

B:

Threads	Time (m-s)	Time (s)
1	0m4.503s	4.503
2	0m2.436s	2.436
4	0m1.636s	1.636
8	0m1.572s	1.572
12	0m1.538s	1.538

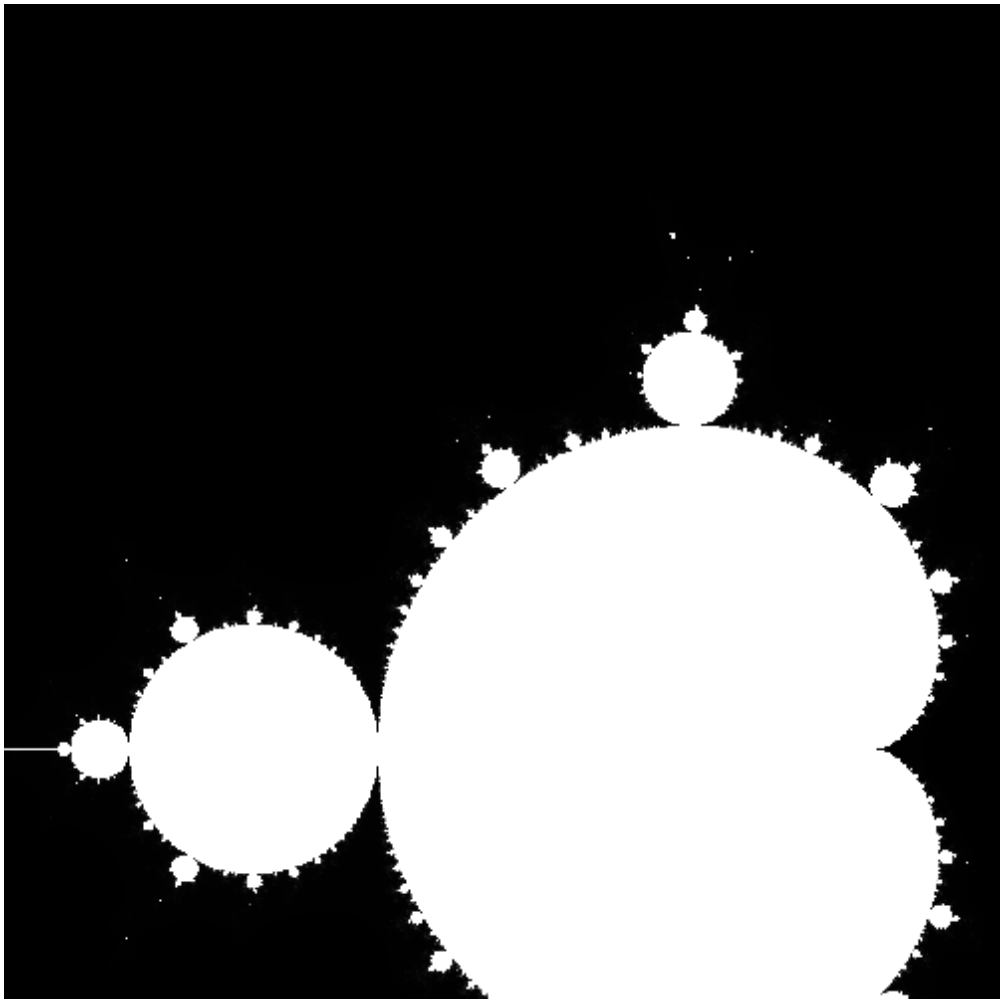


Discussion

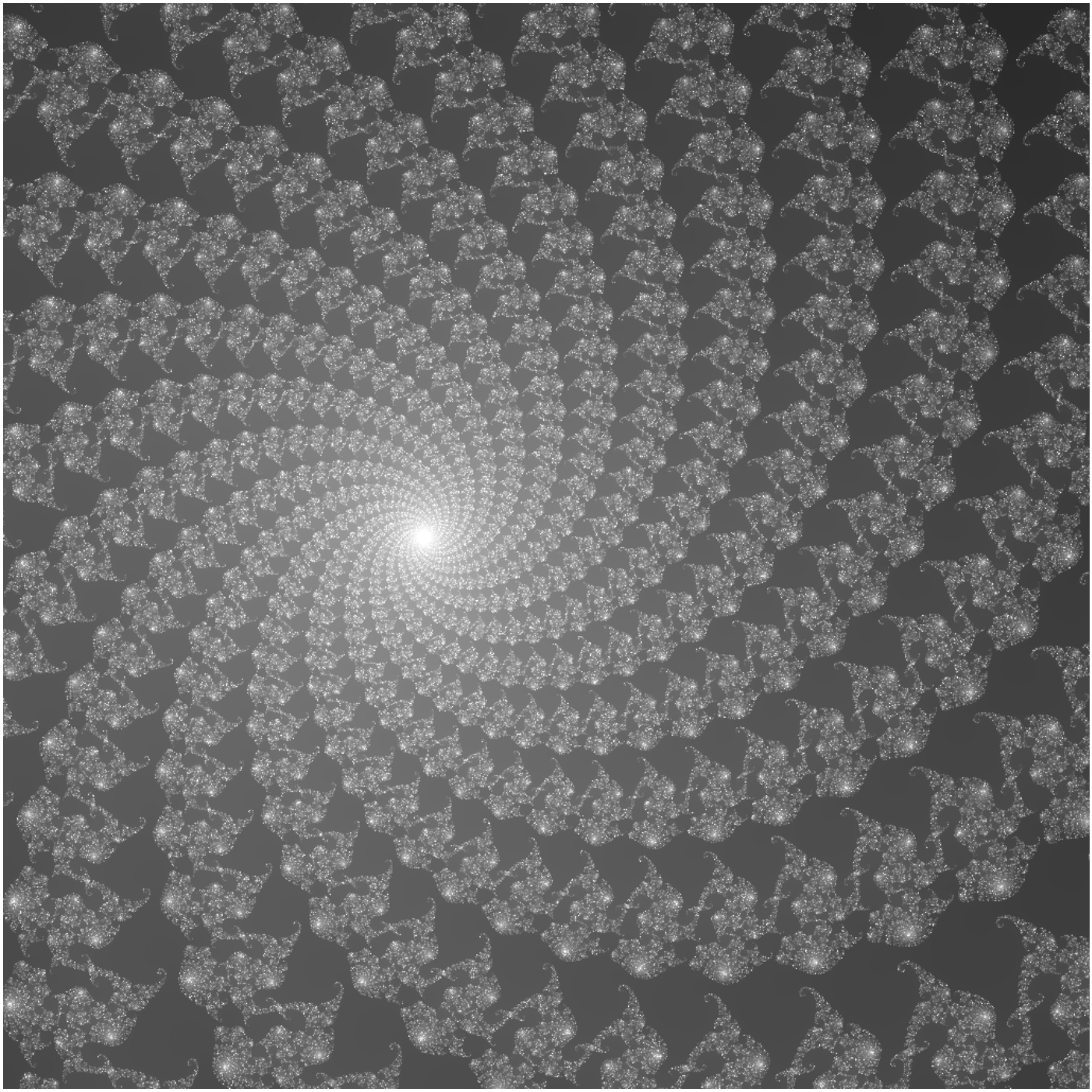
Both curves decrease in execution time from 1 to 4 processes, and adding more threads seems to be not as effective, as the line becomes horizontal, leveling off at around 1.5 seconds for B and 0.7 seconds for A. As such, the optimal number of threads according to the experiments is 4.

Curve B has a much more dramatic drop in execution time from 1 to 2 threads due to the size and complexity of the image being computed, in that the image for B is 1024x1024 pixels in size compared to the image for A which is the default size of 500x500. Additionally, since the image in B is much more zoomed in, at a scale of 0.0000001 compared to 1, the image is much more detailed, which can be seen in the images generated below:

A: `mandel -x -.5 -y .5 -s 1 -m 2000`



B: `mandel -x 0.2869325 -y 0.0142905 -s .000001 -W 1024 -H 1024 -m 1000`



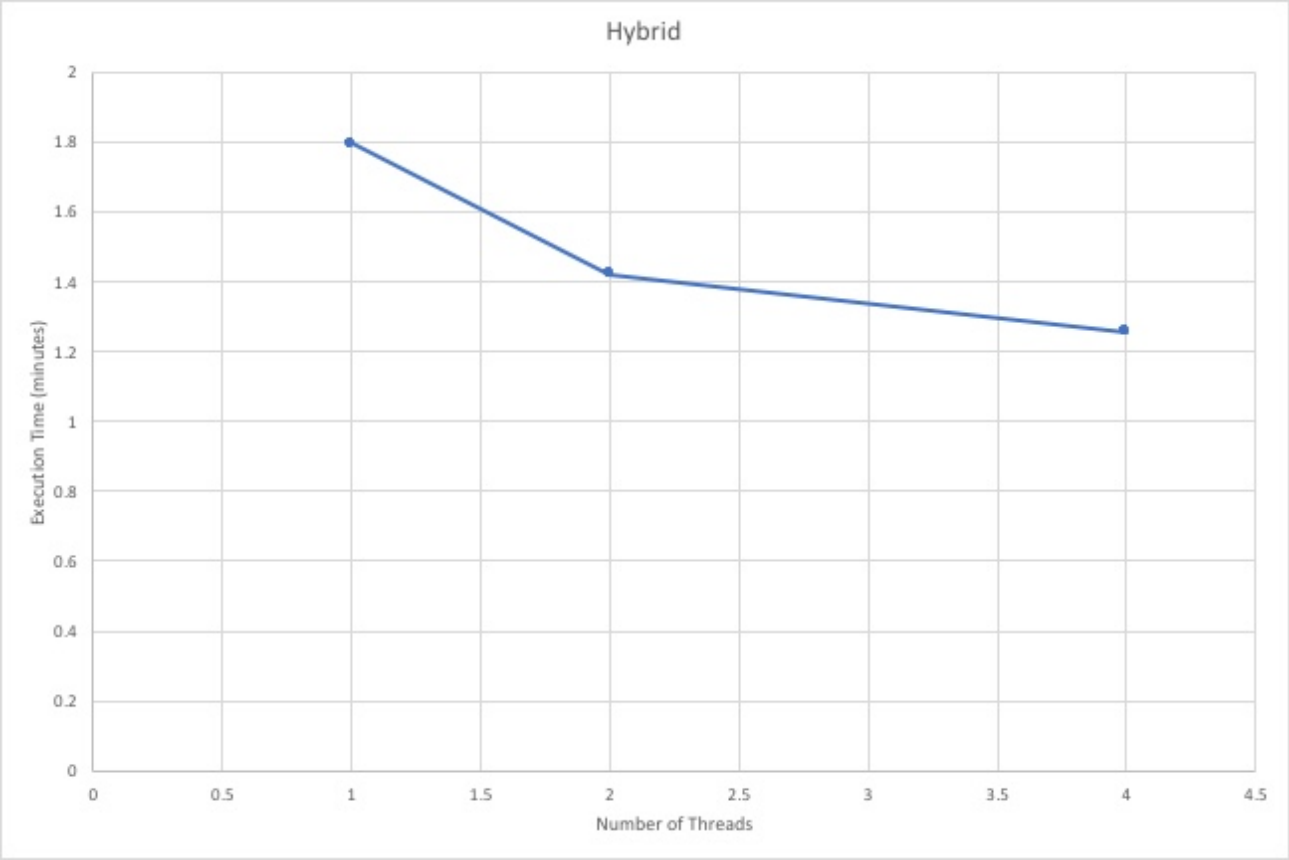
Hybrid

Experimental Setup

The modification to the code in `mandelmovie.c` can be found in the `README.md` file, but involves modifying the function `void buildCommand(char **cmd, double s, int i)` as well as changing the `MIN_ZOOM` value defined at the header of the program from `0.00001` to `0.000001`. The `-n` flag was modified within the `buildCommand()` function as well, so as to change the number of threads the process would spawn.

Graph

Threads	Time (m-s)	Time (m)
1	1m47.793s	1.79655
2	1m25.330s	1.42217
4	1m15.418s	1.25697



Discussion

The execution time did speed up with the more threads that were added, with the graph starting to level off horizontally. If the thread count were to increase to 8, the execution time would probably increase minimally, since the optimal number of threads from previous experiments was 4.

Multithreading and running multiple simultaneous processes have different useful purposes, with threads being useful to speed up individual processes, and having multiple processes running simultaneously being quicker than running them all sequentially. However, since multithreading could replace running multiple processes, while the reverse cannot be done, since data between different processes is not shared, threads are therefore more useful.

The hybrid experiment also showed the effectiveness of multithreading over running simultaneous processes, in that the execution time was able to decrease at a relatively constant rate.